PROGRAMMAZIONE E MODELLAZIONE A OGGETTI - A.A. 2019/2020 - 9 CFU

PROGETTO PIZZERIA ONLINE

Realizzato da:

Nome: Alessio
Cognome: Bernardini
Matricola: #292121

Titolo progetto: Pizzeria Project

1- Specifica Progetto

Realizzare un software grafico per gestire le ordinazioni di una pizzeria online.

Il programma permette all'utente di:

1- Aggiungere menu al suo ordine.
2- Rimuovere menu dal suo ordine.
3- Calcolare alcune statistiche riguardante il suo ordine.
4- Aggiungere i suoi dati personali.
5- Confermare il suo ordine.

Le statistiche di interesse sono:

1- Il prezzo totale dell'ordine, di tutte le pizze e di tutte le bevande ordinate.
2- La quantità dei menu ordinati.
3- Il prezzo medio dei menu, delle pizze e delle bevande ordinate.

Ogni pizza avrà un nome, che identifica il tipo di pizza ordinata (es. margherita, rosmarino…), una lista di ingredienti con cui è realizzata e un prezzo, invece ogni bevanda avrà un nome, che identifica il tipo di bevanda ordinata (es. acqua, coca cola…) e un prezzo.

Il nome e gli ingredienti saranno in formato testuale breve, il prezzo sarà espresso in euro.

I dati personali del cliente richiesti sono:

1- Nome, in formato testuale breve.
2- Cognome, in formato testuale breve.
3- Indirizzo, in formato testuale breve.
4- Città, in formato testuale breve.
5- Numero di telefono, in formato long.

L'ordine del cliente contiene diversi menu, ogni menu contiene una pizza e una bibita che è possibile scegliere tra diverse varietà e il cliente può aggiungere, senza limitazioni, menu al suo ordine.

Il programma deve poter salvare l'ordine in un file di testo, in formato Json. Eventuali release future potranno prevedere l'interfacciamento con un database.

2- Studio del problema.

- Quali sono i punti critici?
- Come si scegli di affrontarli?

I punti critici del programma sono i seguenti:

- L'acquisizione del menu.
- La struttura generale per gestire l'ordine e i menu.
- La costruzione dell'oggetto pizza.
- Il calcolo delle statistiche.
- Acquisizione dati cliente.
- Scrittura sul file di testo in formato json.
- Possibilità di aggiungere un database in futuro.

Per l'acquisizione del menu si è scelto l'inserimento di due combobox, con le quali l'utente può scegliere il tipo di pizza e bibita; tra le diverse modalità di interfacciamento, si è preferito l'uso di combobox per ridurre al minimo lo spazio.

Per gestire la struttura generale dell'ordine, la soluzione adottata è stata utilizzo del Composite pattern, dove gli oggetti composti sono i vari menu ordinati dal cliente, mentre le foglie sono rappresentate dalla pizza e dalla bevanda.

Per costruire l'oggetto pizza ho scelto l'utilizzo del Builder pattern, questo perché la pizza è un oggetto complesso costruita con vari ingredienti.

Il cliente durante la sua ordinazione vedrà comparirsi in una ListView tutti i menu che ha ordinato, questo per agevolare la gestione dall'ordine, infatti grazie a questa lista potrà eliminare menu a suo piacimento o aggiungerne altri, tramite le due combobox trattate sopra.

Aggiungendo o rimuovendo menu all'ordine si aggiorneranno anche le statistiche in basso a destra, queste statistiche, elencate nella specifica, saranno calcolate utilizzando il Visitor Pattern.

Infine, quando il cliente confermerà l'ordine, dovrà prima inserire i suoi dati personali per la spedizione, tramite delle semplici textbox, successivamente, verrà stampato lo scontrino con la possibilità di confermare l'ordine ufficialmente o modificarlo, in caso di ripensamenti.

Se il cliente confermerà l'ordine, quest'ultimo verrà riportato in un file di testo secondo la sintassi del formato json, insieme agli ordini già effettuati.

Visto che la specifica prevede la possibilità di interfacciarsi con un database in futuro, dovrò fare in modo che la classe che gestirà il database (che ora gestisce il file di testo) verrà istanziata una sola volta per non avere conflitti, per assicurarmi ciò, utilizzerò il Singleton Pattern.

3- Scelte architetturali.

- Descrizione dell'architettura software comprensiva di uno schema delle classi UML che descriva le componenti principali del sistema.
- Descrizione e motivazione dei design pattern utilizzati.

Il diagramma delle classi verrà allegato all'interno della cartella "RELAZIONE", questo per le sue dimensioni.

Il design pattern utilizzati sono i seguenti:

- Pattern MVC (Model – View – Controller)
- Pattern Builder
- Pattern Singleton
- Pattern Composite
- Pattern Visitor

Come struttura di base del progetto ho usato il pattern MVC, è un pattern architetturale, in grado di separare la logica di presentazione dei dati dalla logica di business.

Il pattern Model-View-Controller divide la logica del programma in tre parti:

- *Model:* dovrebbe essere responsabile per i dati del dominio dell'applicazione
- *View:* presenta la visualizzazione del modello nell'interfaccia utente.
- *Controller:* è davvero il cuore di MVC, l'intermediario che lega insieme il Modello e la Vista, ovvero prende l'input dell'utente, manipola il modello e fa aggiornare la vista.

L'idea è quella di separare l'interfaccia utente in View (crea il display, chiamando i Model quando è necessario per ottenere informazioni) e Controller (risponde alle richieste dell'utente, interagendo con View e Controller se necessario).

Tutti i componenti del pattern sono separati con la propria funzionalità, in altre parole, il modello MVC consiste nel suddividere il comportamento dell'interfaccia utente in parti separate al fine di aumentare le possibilità di riutilizzo e la testabilità.

Proprio per questi suoi punti di forza come la possibilità di riutilizzo e testabilità ma anche il fatto di semplificare le operazioni successive di mantenimento e aggiornamento che ho deciso di applicare questo pattern.

Per gestire la struttura dell'ordine e dei vari menu contenuti in esso, come accennato prima, si è scelto di utilizzare il pattern composite.

Ho deciso di utilizzare questo pattern perché permette di trattare un gruppo di oggetti come se fossero l'istanza di un oggetto singolo, perfetto per gestire l'ordine.

Il pattern composite organizza gli oggetti in una struttura ad albero, nella quale i nodi sono degli oggetti composti, che sono rappresentati dai vari menu che contengono una pizza e una bibita che rappresentano le fogli dell'albero.

La bibita sarà una semplice classe, invece la pizza verrà costruita sfruttando il pattern builder.

E' un pattern di tipo creazionale, il suo scopo principale è quello di separare la costruzione di un oggetto complesso dalla sua rappresentazione cosicché il processo di costruzione stesso possa creare diverse rappresentazioni.

Ciò ha l'effetto immediato di rendere più semplice la classe pizza, permettendo a una classe builder separata di focalizzarsi sulla corretta costruzione di un'istanza e lasciando che la classe originale si concentri sul funzionamento degli oggetti.

Per cui avrò la classe Concretebuilder che costruisce e assembla gli ingredienti della pizza implementando l'interfaccia IBuilder, definendo e tenendo traccia della rappresentazione che crea, la classe Cook (cuoco) che costruisce/"cucina" l' oggetto pizza utilizzando l'interfaccia IBuilder e la classe Pizza che rappresenta l'oggetto complesso e include i suoi ingredienti in una lista.

Per calcolare le statistiche, ho utilizzato il pattern visitor, è un pattern comportamentale, che permette di separare degli algoritmi dalla struttura di oggetti composti a cui è applicato, in modo da poter aggiungere nuove operazioni e comportamenti senza dover modificare la struttura stessa.

Per cui la mia struttura complessa è l'ordine del cliente, gestito tramite il pattern composite, invece gli algoritmi applicati su essa, mi calcoleranno:

1 - Il prezzo totale dell'ordine, di tutte le pizze e di tutte le bevande ordinate.
2 - La quantità dei menu ordinati.
3 - Il prezzo medio dei menu, delle pizze e delle bevande ordinate.

Ho scelto di sfruttare questo pattern appunto per separare la complessità di calcolo di questi valori e per semplificare l'aggiunta di altre statistiche utili, che potranno essere aggiunte in futuro.

Infine, il pattern singleton, è un pattern di tipo creazionale, che ho utilizzato per la costruzione della classe DBHendler, ovvero la classe che gestisce la scrittura sul file di testo e che in un futuro un eventuale database.

Ho deciso di utilizzare questo pattern, perché dovevo essere sicuro che venisse creata una sola istanza di questa determinata classe, in caso contrario potrei riscontrare dei conflitti tra le classi che mi gestiscono il database che sarà uno solo.

4- Documentazione sull'utilizzo.

- (Se applicabile) Con quali parametri va eseguito il software, una volta compilato?
- (Se applicabile) Ci sono passi particolari da eseguire per la compilazione?

Il programma prevede la possibilità di riportare gli ordini in un file di testo, in formato json.

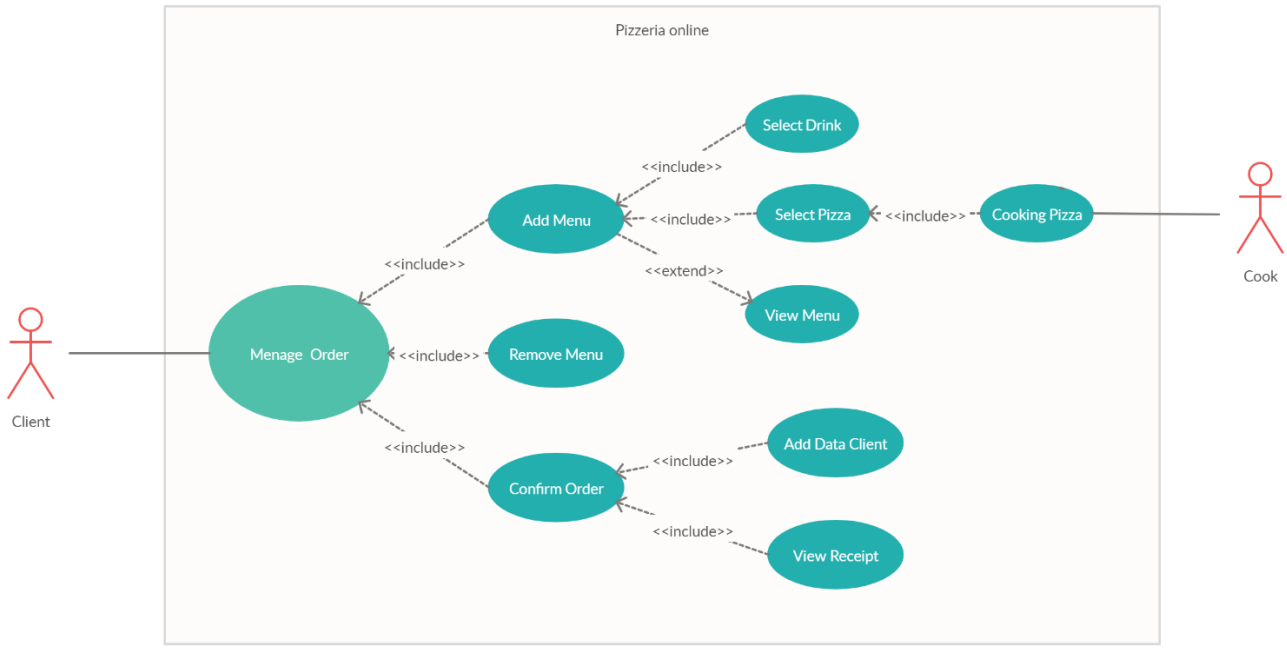
Il file di testo si chiama "db2" e si trova all'interno della cartella "File Json", per far si che il programma riesca a scrivere su questo file è obbligatorio passargli il percorso del file di testo, bisognerà specificare questo all'interno della classe "Program.cs", dentro la cartella "UseMVCApplication".

Un percorso assoluto è definito tale quando specifichi la posizione di un elemento a partire dalla radice del file system. Esso non è dipendente dalla cartella di lavoro corrente.

5- Use Cases con relativo schema UML.

- Descrivere i più significativi.

Lo diagramma Use Cases è il seguente:



In seguito riporto la descrizione dei casi d'uso più significativi, ovvero "Add menu", "Remove menu" e "Confirm order" :

Use Case: Add menu

Id: UC1

Actor: Client

Preconditions:

- Selezionare una pizza dalla combobox
- Selezionare una bevanda dalla combobox

Basic course of events:

1. Cliccare sul bottone "Aggiungi Menu"
2. Creazione oggetto Pizza e Drink
3. Creazione del Menu e aggiunta del Menu nell'ordine
4. Aggiunta del menu nella ListView

Postconditions:

- Aggiunta del menu nell'ordine finale
- Apparizione del menu nella ListView

Alternative paths:

5. Se la pizza e la bevanda non sono selezionate correttamente apparirà un messaggio d'errore

Use Case: Remove Menu

Id: UC2

Actor: Client

Preconditions:

- Almeno un menu deve essere selezionato nella listview

Basic course of events:

1. Click sul bottone "Rimuovi Menu"
2. Il sistema ricerca i menu selezionati nell'ordine
3. Rimozione dei menu dall'ordine
4. Rimozione dei menu dalla listview

Postconditions:

- I menu selezionati saranno eliminati dall'ordine e dalla listView
- Il cliente potrà aggiungere altri menu o confermare l'ordine

Alternative paths:

5. Se nessun menu è stato selezionato il bottone non farà nulla

Use Case: Confirm Order

Id: UC3

Actor: Client

Preconditions:

- Almeno un menu aggiunto all'ordine

Basic course of events:

1. Click sul bottone "Conferma Ordine"
2. Compilazione Dati cliente e creazione oggetto cliente
3. Stampa dello scontrino
4. Scelta Conferma o modifica ordine finale

Postconditions:

- Nel caso in cui il cliente abbia scelto di confermare l'ordine verrà salvato in file json e verrà visualizzato un messaggio di ringraziamento
- Nel caso in cui il cliente abbia scelto di modicare l'ordine ritornerà nell'interfaccia principale

Alternative paths:

5. Nel caso in cui l'ordine non contenga neanche un menu verrà stampato un messaggio d'errore